Computer Networks 168 (2020) 107040

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Forecasting assisted VNF scaling in NFV-enabled networks

Yifu Yao[a], Songtao Guo[b,*], Pan Li[a], Guiyan Liu[a], Yue Zeng[a]

[a] Chongqing Key Lab of Nonlinear Circuits and Intelligent Information Processing, College of Electronic and Information Engineering, Southwest University, Chongqing, 400715, China
[b] Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China, and College of Computer Science, Chongqing University, Chongqing 400044, China

A R T I C L E I N F O

Article history:
Received 6 May 2019
Revised 3 November 2019
Accepted 26 November 2019
Available online 29 November 2019

Keywords:
Network function virtualization
Forecasting mechanism
Service chains
Cost minimization
System performance

A B S T R A C T

As a new promising technology, Network Function Virtualization (NFV) converts hardware based network function into software module running on virtual machines, for cost reduction and ease of management. These virtualized networks functions (VNFs) are commonly organized together as service function chains. Properly deploying VNFs is a key to achieve NFV targets. Most of existing efforts focus on one-time placement, ignoring the dynamic deployment and scaling needs of VNFs for the time-varying system. In this paper, we study the dynamic deployment and scaling of VNFs for operation cost minimization. We first formulate an offline VNF deployment cost minimization problem and prove its NP-hardness. Then, considering the dynamics of the network, we propose an efficient online scaling algorithm, which is composed of two parts: 1) One is Fourier-Series-based forecasting approach to minimize cost by avoiding frequent changes in network topology and 2) the other is online deployment algorithm to properly deploy VNF instances. We finally evaluate the proposed algorithms and results show that our algorithms can reduce more than 20% cost while maintaining the same system performance as other heuristic algorithms.

1. Introduction

Modern communication network attaches great importance to data transfer security, resource utilization and system performance. These concerns used to be solved by dedicated hardware middleboxes, such as firewalls, network address translators (NATs) and deep packet inspection (DPI), which are costly and difficult to reconfigure [1]. Network Function Virtualization (NFV) as an emerging technology revolutionizes the handling of network functions, which applys virtualization technology to consolidate network functions onto industry-standard virtualized platforms, e.g., virtual machines (VMs) on high-volume servers, aims at achieving configuration and scaling flexibility, as well as cost minimization [2,3].

The network containing VMs is referred to as NFV-Enabled network. Data in such network are transmitted in the form of service chain, which consists of a sequence of network functions in a specified order [4,5]. In practice, the NFV-Enabled network is a time-varying system and needs to be constantly redeployed by adding or removing VMs with the change of demands [6]. How to dynamically deploy virtual network functions (VNFs) is a key to achieve NFV targets. A concern on VNF deployment problem is how to reduce deployment cost to increase revenue for service providers. In the meantime, quality of service (QoS) is another concern in VNF deployment. Due to the sharing nature of data centers, there may exist some other workloads called background traffic in the network when processing network functions, which may occupy bandwidth resources [7]. Ignoring the background traffic when deploying the VNFs may cause unexpected performance degradation due to link congestion. These concerns motivate us for this work on the dynamic deployment and scaling of VNFs.

However, there exist some challenges for the dynamic deployment and scaling of VNFs. First, the demands of traffics may change from time to time. This requires us to consider a time-varying system where all requests are dynamically generated. Second, the deployment of VNFs determines the routing path of each request, which further influences the quality of network service. Third, an online routing algorithm is needed to process flows to minimize the transmission cost while satisfying the delay constraint. To address these challenges, some efforts have been taken place in recent years. Duan et. al. designed a system to manage scaling of VNF service chains in control plane and data plane [8]. They proposed that service chain scaling is a good way to achieve NFV goals and combined reactive scaling with proactive scaling. However, the sharing nature of data centers was ignored. Rost and Schmid consolidated all network functions into one data center,

* Corresponding author.
E-mail address: songtao_guo@163.com (S. Guo).

https://doi.org/10.1016/j.comnet.2019.107040
1389-1286/© 2019 Elsevier B.V. All rights reserved.

called middlebox [9]. They figured out a cost-inefficient path but did not consider the order of network functions. Sun et. al. introduced a forecast approach to update network topology [10], but they did not take QoS into account. Jia et. al. found the optimal placement and routing path by constantly changing network topology [11], which was costly and unrealistic under actual situation. What's more, some existing works [12–14] proposed a one-time placement method but ignored the dynamic deployment and scaling needs of VNFs.

In this paper, we consider the horizontal scaling of VNFs and assume the network is operated within service provisioning time s, which is divided into a large span of time slots $1, 2, \ldots, T$. There will be a prediction stage in each time slot. In each prediction stage, we introduce a forecast mechanism based on Fourier-Series to judge whether there are new demands generated in the next time slot. If yes, we will update network topology to serve it. Here, the demands are defined as new network functions that need to be processed. Similar to [15], we take the bandwidth usage for other workloads as a time-varying pattern. In fact ignoring these workloads may cause link congestion. To maintain the system performance, we adopt an online learning framework to propose our deployment algorithm and construct a multi-level acyclic graph to ensure the order of processing.

Compared to the existing works [12–14,16–19] our work has the following advantages: (i) A time-varying model is constructed and the dynamic deployment and scaling need of VNFs is considered; (ii) Energy consumption is reduced while system performance is taken into account, which is reflected by end-to-end delay and link congestion. In summary, the main contributions of this paper can be summarized as follows.

- We construct a VNF placement model that contains the dynamic creation of VNFs and dynamic routing between different VNF instances. The aim of the model is not only reducing VNF placement cost but also improving the quality of service in terms of end-to-end delay and link congestion.
- We propose a forecasting method, which can avoid frequent updates to network topology. This algorithm can reduce the cost and delay of creating VNFs.
- We propose a VNF deployment algorithm based on online learning to eliminate the influence of background traffic. In the meanwhile, we analyze the performance of this algorithm and give its superiority.
- Finally, experimental results show that the proposed algorithms can reduce the cost by 20% while maintaining the performance of the system than the existing works.

The remainder of this paper is organized as follows. Section 2 discusses the related work for NFV management and Section 3 introduces the system model and formulates the optimization problem. Then, Section 4 describes the forecast algorithm to predict demand changes and an online-learning based VNF deployment algorithm is proposed. A routing algorithm for the request in newly network topology is given in Section 5. Finally, we present evaluation results in Section 6 and conclude this paper in Section 7.

2. Related work

In this section, we review some existing works in deployment and scaling of NFV.

2.1. Deployment of NFV

The concern of NFV was first presented from a white paper [2] in which telecommunication operators claimed that network functions could be virtualized and processed on the virtual platform. Early work focused on virtualizing network functions on specialized hardware platforms and set industry standards for the operation of VMs [20–22].

In recent years, there are several studies on NFV deployment. Rost et al. [9] considered the placement of a middlebox in a n-nodes network, through which each flow from source to destination should go. Under the assumption that each middlebox could serve k flows and each flow could only take one network function, they introduced an approximation algorithm with approximation ratio $O(\log \min\{n, k\})$. However, this assumption is too simple and does not in line with the actual situation. M. Jia et al. [13] introduced an auxiliary directed acyclic graph and a heuristic algorithm to minimize the cost of routing with and without end-to-end delay constraints. They assumed VMs could be dynamically added in any data center. In this case, one does not need to choose the VM, which is far from the source-destination pair. But that is just a framework, and the authors do not show how to add VMs specifically.

VNF-P [23] constructed an optimization model to combine the hardware platform with VNFs and considered the end-to-end delay to guarantee QoS. Similarly, Bari et al. [24] studied the optimization problem that minimized the operating cost. However, these studies [23,24] only deal with one-time placement or resource allocation and ignore the dynamics of network. Pham et al. [25] adopted the matching theory and Markov approximation approach to solve VNF placement problem, for the aims of cost minimization. To cope with the VNF many-to-one matching game, they introduced a centralized solution and a distributed solution. Cziva et al. [26] formulated the Edge VNF placement problem to minimize the delay between users and VNFs. In their system model, the delay of network was changing from time to time and the user mobility patterns and QoS were taken into account. With the help of queuing theory, the authors in [27] sought to solve the VNF placement and resource allocation problem. What's more, they also proposed an efficient solution strategy called MaxZ to reduce the solution complexity.

2.2. Scaling of VNF

There are also some studies focused on the scaling of VNFs in recent years. Wang et al. [28] introduced a VNF scaling algorithm based on a classical online ski-rental algorithm. The algorithm used past information as input to get the number of VMs to be needed in the next time slot. Similarly, Sun et al. [10] proposed a forecast-assisted VNF scaling algorithm to update network topology on demand, but QoS was not taken into account. Jia et al. [11] leveraged a regularization method from online learning to divide the optimization problem into a sequence sub-problems, which could decouple the decisions in time slots t and $t-1$. In their system model, end-to-end delay was quantified as part of the cost. But the work in [11] was based on the assumption that there was no mutual interference between each data center. This implies that it does not need to consider background traffic, which violates the sharing nature of data centers.

Rahman et al. [29] introduced a scaling algorithm based on Machine Learning (ML). The proposed ML-based algorithm used past VNF scaling decisions and traffic characteristics as input to generate deployment solution in advance. Toosi et al. [30] combined the horizontal and vertical scaling of VNFs to reduce the cost and the service level agreement in a time-varying system. Duan et al. [8] used the case of IMSs and designed a scaling system for the service chains in control plane and data plane. The authors in [31] first formulated a traffic forecasting algorithm to estimate the upper bound of traffic in a short term. Under this forecasting, two

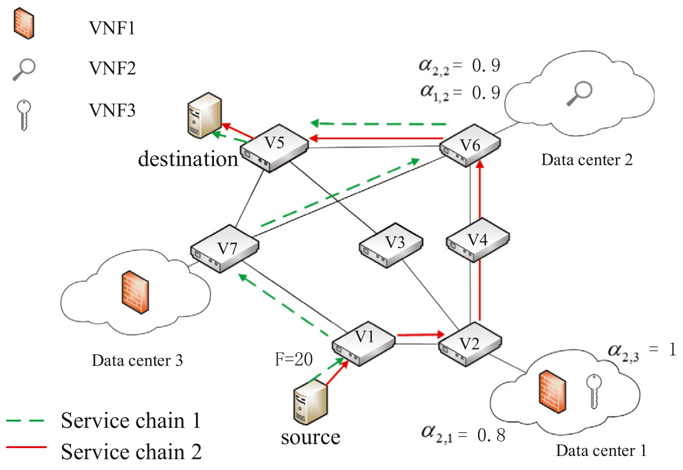


Fig. 1. An example of a network $G(V, E)$ with a switch nodes set V={v_1, v_2, v_3, v_4, v_5, v_6, v_7} and $V_d = \{v_2, v_6, v_7\}$. There are two service chains from source to destination in a specified order.

VNF placement algorithms were proposed to meet the demand of VNF scaling.

Compared with existing works, we consider the dynamic deployment and scaling need of VNFs as well as background traffic. We propose a forecasting mechanism to avoid the frequent changes of network topology and a VNF placement algorithm to eliminate the effects of background traffic.

3. System model and problem formulation

In this section, we first introduce the system model and then present the objective function in offline mode.

3.1. System model

The network topology is represented by a directed graph $G = (V, E)$, where V is the set of switch nodes and E is the set of links. In order to assemble service chains to meet the demand of flows, some nodes that hold VMs to process network functions are attached to data centers. We use V_d to represent the set of nodes connected to data centers. For those nodes that are not attached to data centers or belong to the set V_d but do not perform network functions, we treat them as switch nodes. Notice that compared to the communication cost and delay between nodes in the network, the cost and delay between nodes and data centers can be negligible as they are connected by high-speed optical fiber [32].

The VMs on data centers may run different network functions, which are denoted as VNF instances. We assume that there are a total of M types of VNF instances and I data centers. Since the diversity of CPU, memory, etc., different data centers have different processing capacity for VNF instances [33]. In this paper, we denote the available amount of resource in data center m and instance i by w^r_m, $w^r_{m,i}$ ($r \in R$) respectively, where R represents different types of resources (CPU and memory). In each time slot, there are P traffic flows, and each flow $p \in P$ travels from source s_p to destination d_p to process network functions in a specified order. We use Fig. 1 as an example to demonstrate the network, where switch nodes 2, 6 and 7 are connected to the data center. The definition of other parameters used in this paper are listed in Table 1.

In the following we consider the processing of service chains. Suppose that a flow is denoted by $f_p = (s_p, d_p, b_p, sc_p)$, where s_p, d_p indicate the source and destination, respectively, b_p is the requirement of bandwidth and sc_p represents the service chain. For ease of modeling, we use $sc_p = \{sc_{p,1}, sc_{p,2}, \ldots, sc_{p,m}\}$ to indicate different network functions. Without loss of generality, the flow rate may change after passing through some VNF instances [34]. For example, the network function such as video transcoding will change the format of data packets, resulting in a change in the flow rate; security functions (e.g. deep packet inspection, firewall) may drop some packets that do not comply with security policy. Thus we use $F_{p,v,v'}$ to indicate the flow rate of p-th flow between two adjacent nodes v and v', use $\alpha_{p,m}$ to represent the change rate of flow p passing through m-th VNF instance. Notice that we can get $\alpha_{p,m}$ by estimating and calibrating over time. We can calculate the flow rate of flow p after passing through the m-th VNF by

$$\overline{F_{p,v,v'}} = F_{p,v,v'}\alpha_{p,m} \tag{1}$$

As shown in Fig. 1, we suppose the initial flow rate of service chain 2 is 20. When flow p passes through V_2, the flow rate changes correspondingly. From (1) we can get $\overline{F}_{2,2,4} = 12$ (since $\alpha_{2,1} = 0.6$, $\alpha_{2,3} = 1$) and $\overline{F}_{2,6,5} = 10.8$ (since $\alpha_{2,2} = 0.9$). Notice that if service chain 2 only processed VNF3 while passing through data center 1, then $\overline{F}_{2,2,4} = 20$.

In each service provisioning time, we introduce logic link $x^{(t)}_{p,m,i,m',i'}$. The $x^{(t)}_{p,m,i,m',i'}$ is a non-negative real number and represents the flow rate of p-th flow from VNF instance m in data center i to VNF instance m' in data center i'. $band_{i,i'}$ is the bandwidth capacity constraint at logic link ii', which is equal to the minimum value in physical links.

3.2. Cost function construction

The processing of network functions in data centers will cause cost and delay. Similarly, we assume the transmission of flow also causes cost and delay. Given a flow p going through data centers as its specified order from source to destination, our goal is to minimize the total cost of the network. The overall cost can be divided into two parts. One is the deployment cost $C^{(t)}_D$ and the other is the implementation cost $C^{(t)}_I$, which contains transmission and processing cost.

$$C^{(t)} = C^{(t)}_D + C^{(t)}_I \tag{2}$$

3.2.1. Deployment cost

Creating a new VNF instance typically requires transferring the image of the VM to the data center and attach the image to the device. Without loss of generality, we use $\beta_{m,i}$ to represent the cost of deploying a VNF instance m in data center i and use $d_{m,i}$ to denote the delay of deploying a VNF instance m in data center i. Due to that the cost caused by the removal of VNF instance is relatively small, we regard the creation of VNF instance as the main cost [35] and the delay is the same. Let $\delta^{(t)}_{m,i}$ denote the number of creating new VNF instances m in data center i, which can be obtained by

$$\delta^{(t)}_{m,i} = \max\{0, \rho^{(t)}_{m,i} - \rho^{(t-1)}_{m,i}\} \tag{3}$$

where $\rho^{(t)}_{m,i}$ is the number of VNF instances m contained in data center i in time slot t. Thus we can calculate the total cost of deployment by

$$C^{(t)}_D = \sum_{i\in[I]}\sum_{m\in[M]} \beta_{m,i}\delta^{(t)}_{m,i} \tag{4}$$

Similarly, the total delay of VNF deployment can also be obtained by (5)

$$d^{(t)}_D = \sum_{i\in[I]}\sum_{m\in[M]} d_{m,i}\delta^{(t)}_{m,i} \tag{5}$$

3.2.2. Implementation cost

In this section, we consider the cost of transmission and processing of network functions in data centers.

Table 1
Notations.

Parameters	Description
P	Number of flows
M	Number of VNF types
I	Number of data centers in system
R	Types of resources of a PM
X	Set of feasible solution
$\rho_{m,i}^{(t)}$	Number of VNF m contained in data center i
$\delta_{m,i}^{(t)}$	Number of new VNF m deployed in data center i
s_p, d_p, b_p	Source, destination and bandwidth requirement of flow p
$x_{p,m,i,m',i'}^{(t)}$	Incoming flow rate of flow p from VNF m in data center i to VNF m' in data center i'
$F_{p,v,v'}$	Flow rate of p-th flow between node v and v'
$\alpha_{p,m}$	Change ratio of flow p after passing through VNF m
$r_{j,t}$	Reward of arm j in time slot t
$\bar{r}_{j,t}$	Average reward of arm j during t
$\bar{r}_t^{\max}$	Average reward of optimal arm during t
$c_{T,n_j(t)}$	Confidence interval
$n_j(t)$	Number of times that arm j is selected during time t
$n^{\max}(t)$	Number of times that optimal arm is selected during time t
μ_j	Reward expectation of arm j
$\mu^{\max}$	Greatest reward expectation of all arms
$a_{j,t}$	Indicator function
$w_{m,i}^r$	Capacity of VNF m in data center i for resource type r
w_i^r	Capacity of data center i for resource type r
$band_{i,i'}$	Capacity of logic link ii'
$d_{m,i}$	Delay of deploying a new VNF m in data center i
$\varphi_{p,i,i'}$	Delay of transmission from data center i to data center i'
λ_m	Delay of processing VNF m
$\beta_{m,i}$	Deployment cost of VNF m in data center i
$\gamma_{p,i,i'}$	Transmission cost of p-th flow from data center i to i'
τ_m	Processing cost of VNF m

As we know, the transmission of data on link is often accompanied by the cost of energy. Here we let $\gamma_{p,i,i'}$ indicate the cost of sending a unit of data from data center i to i' by the p-th flow on logical link. Recalling the flow rate defined, we can get the overall cost of transmission as

$$C_l^{(t)} = \sum_{p\in[P]}\sum_{m\in[M]}\sum_{m'\in[M]}\sum_{i\in[I]}\sum_{i'\in[I]} x_{p,m,i,m',i'}^{(t)}\gamma_{p,i,i'} \tag{6}$$

Notice that if different VNF instances are in the same data center, then the cost of transmission is 0. Similarly, we use $d_l^{(t)}$ to represent transmission delay, which can be computed by

$$d_l^{(t)} = \sum_{p\in[P]}\sum_{m\in[M]}\sum_{m'\in[M]}\sum_{i\in[I]}\sum_{i'\in[I]} x_{p,m,i,m',i'}^{(t)}\varphi_{p,i,i'} \tag{7}$$

where $\varphi_{p,i,i'}$ represents the delay caused by the p-th flow sending a unit of data from data center i to i'.

In the following we consider the processing cost of network functions in data centers. In a time slot, we assume that there are P flows and each flow contains $|sc_p|$ network functions that need to be processed. Let τ_m be the cost of processing a VNF instance m and the overall cost is given by

$$C_r^{(t)} = \sum_{p\in[P]}\sum_{m\in[M]} \tau_m|sc_p| \tag{8}$$

Furthermore we can also get the delay of processing VNFs in (9) by introducing the delay of processing a VNF instance m.

$$d_r^{(t)} = \sum_{p\in[P]}\sum_{m\in[M]} \lambda_m|sc_p| \tag{9}$$

3.3. Offline cost minimization problem

In offline model, we assume the information of the network is known and the optimization problem can be obtained in (10).

$$\min \sum_{t\in[T]} C_D^{(t)} + C_l^{(t)} \tag{10}$$

subject to

$$\sum_{p\in[P]} x_{p,m,i,m',i'}^{(t)} \le \rho_{m,i}^{(t)} w_{m,i}^r \quad \forall t\in[T], i\in[I], i'\in[I], m\in[M], m'\in[M], r\in[R] \tag{11a}$$

$$\sum_{p\in[P]} F_{p,v,v'} \le w_i^r - \sum_{m\in[M]} w_{m,i}^r \quad \forall i\in[I], v, v'\in[V], \forall m\in[M], r\in[R] \tag{11b}$$

$$\sum_{m\in[M]} w_{m,i}^r \le w_i^r \quad r\in[R], i\in[I] \tag{11c}$$

$$\delta_{m,i}^{(t)} \ge \rho_{m,i}^{(t)} - \rho_{m,i}^{(t-1)} \quad \forall t\in[T], m\in[M] \tag{11d}$$

$$\sum_{p\in[P]} b_p \le band_{i,i'} \quad \forall p\in[P], i, i'\in[I] \tag{11e}$$

$$d_D^{(t)} + d_l^{(t)} + d_r^{(t)} \le d^{(t)} \quad \forall t\in[T], i\in[I], m\in[M] \tag{11f}$$

$$\delta_{m,i}^{(t)} = \{0,1,2,\ldots\} \quad \forall t\in[T], i\in[I], m\in[M] \tag{11g}$$

$$\rho_{m,i}^{(t)} = \{0,1,2,\ldots\} \quad \forall t\in[T], i\in[I], m\in[M] \tag{11h}$$

Here $d^{(t)}$ is the threshold of delay. Constraints (11a) and (11b) ensure that there are sufficient resources on the data center i and VNF instance m to provide service. Constraint (11c) guarantees that the total sum of capacities of VNFs on data center i cannot exceed the maximum capacity of i. Inequality (11e) is the link bandwidth constraint. Constraints (11d) and (11g) are originated from (3) that the number of new VNF instances is an integer and we only consider the creation of VNF instances. The total system delay is defined in (11f), which cannot exceed the threshold $d^{(t)}$.

Theorem 1. *The offline optimization problem (10) is NP hard.*

Proof. We can reduce the problem (10) to a special case of delay-constrained shortest path problem, which is known to be NP-hard. Under restrictions of some demands, our goal is to find a path with minimum cost while satisfying the delay constraints. We assume the cost and delay of deployment and processing is 0, that is $C_r^{(t)} = C_D^{(t)} = 0,\ d_r^{(t)} = d_D^{(t)} = 0$. The cost of system is represented by transmission cost $C_l^{(t)}$ and the weight is $\gamma_{p,i,i'}$. According to constraint (11e) we know the required demand, so the problem (10) is equivalent to delay-constrained shortest path problem. □

Different from the offline problem, the online routing path and VNF instances will vary according to different flows. We seek to introduce an online scaling algorithm to dynamically scale the VNF with minimum cost. The solution to the online scaling problem can be divided into two steps. First, forecasting the VNF instances in the subsequent time slot to determine whether there are new demands generated. If yes, we will adopt a deployment algorithm to reconfigure the network topology. Second, we will introduce a routing algorithm to find a routing path for each request.

4. Online forecast-assisted optimization problem

It is known from (10) that the whole cost contains deployment and implementation cost. According to Sun et al. [10], the initial cost of deploying VNF instances is often larger. To achieve the goal of minimum cost, it is a good way to avoid frequent updates to network topology. In this paper, we update the network topology according to whether there are new VNF demands generated in the next time slot. Thus we would like to propose a forecast mechanism to help us make decision.

4.1. Online forecast algorithm

Due to that redeploying VNF instances is costly and time-consuming, service providers may prefer to predict the change in demands and update the network topology correspondingly. It is known from [10] that the number of newly added VNF instances can be obtained by the change of flow rates. Thus, our problem can be reduced to predicting the flow rate in the next time slot based on historical information. The basic idea of predicting in our work is to use the Fourier-series-based method for nonlinearly fitting. We let $Y(t)$ denote the flow rate of previous n time slots. By the *Fourier* transform, we can get

$$Y(t) = a_0 + \sum_{k=1}^{m}[a_k \cos(k\omega t) + b_k \sin(k\omega t)] \tag{12}$$

where a_0, a_k, b_k is the *Fourier* coefficient. However, the traffic characteristic in the NFV-enabled network changes from time to time, and the coefficients may not work in different situations. Accordingly, we introduce trend function $Tr(t)$ and the training data can be formulated as the following regression model

$$\mathbf{Y} = \mathbf{AX} + \mathbf{B} \tag{13}$$

where

$$\mathbf{Y} = [Y(t_1), Y(t_2), \ldots, Y(t_n)]^T$$

$$\mathbf{X} = [a_0, a_1, b_1, a_2, b_2, \ldots, a_m, b_m]^T$$

$$\mathbf{B} = [Tr(t_1), Tr(t_2), \ldots, Tr(t_n)]$$

$$\mathbf{A} = \begin{bmatrix} 1 & \cos(\omega t_1) & \sin(\omega t_1) & \cos(2\omega t_1) & \sin(2\omega t_1) & \ldots & \sin(m\omega t_1) \\ 1 & \cos(\omega t_2) & \sin(\omega t_2) & \cos(2\omega t_2) & \sin(2\omega t_2) & \ldots & \sin(m\omega t_2) \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & \cos(\omega t_n) & \sin(\omega t_n) & \cos(2\omega t_n) & \sin(2\omega t_n) & \ldots & \sin(m\omega t_n) \end{bmatrix}$$

It is well known that *Fourier* coefficients can be calculated by the least square method ($\mathbf{X} = (A^TA)^{-1}A^TY$) and the accuracy can be improved by adjusting the fitting order (i.e., m in (12)) according to the determinable coefficient $R^2 = \frac{\sum_{i=1}^{n} Y(i) - \hat{Y}(i)}{\sum_{i=1}^{n} Y(i) - \bar{Y}}$, where $\hat{Y}(i)$ is the fitted value and $\bar{Y}$ is the average of the training set.

It is worth noting that the regression model may underestimate the flow rate injected into the network. In this case, the network cannot handle all requests, which impacts service availability. To solve this problem, we should reserve some redundant capacity to make sure the network has enough resources in the case that the actual incoming traffic is greater than the estimated one [31]. Hence, we adopt $3-\sigma$ principle to ensure the network has enough resources, where σ is the standard variance of the estimated and real flow rate in the training set. Based on this principle and regression model, the predicated value can be obtained as

$$\hat{Y}(t) = Y(t) + 3\sigma + Tr(t) \tag{14}$$

As discussed above, the number of newly added VNF instances $\Delta\rho_m^{(t)}$ can be obtained by the change of flow rate. In the next section, we will study how to deploy these instances properly.

4.2. Online VNF placement algorithm

The deployment of VNF instances determines the routing path, which further influences the routing and processing cost. However, due to the existence of background traffic, blindly seeking the minimum cost may cause link congestion. How to deploy the VNF instance appropriately is a key problem to be addressed in this section.

To maintain the system performance, we use $l_{i,i'}^{(t)}$ to indicate the background traffic on logic link ii' and use the maximum flow rate on the link as the objective function.

$$\min \max_{i,i' \in I} \sum_{m,m' \in [M]} \sum_{p \in [P]} (x_{p,m,i,m',i'}^{(t)} + l_{i,i'}^{(t)}) \tag{15}$$

subject to constraints (11a)–(11f)

Different from previous studies [13,36], which assumed that service providers can obtain the whole information on each link, in our system, we can get the background traffic from i to i' if and only if the flow goes through the link ii'. Due to the noise and randomness of background traffic, the background traffic on links is a time varying pattern and obeys the random distribution with an unknown expectation [28], which is similar to the multi-armed bandit problem (pull the arm of one of the machines and receive a random reward). Thus, we prefer to design a multi-armed bandit algorithm based on online learning to solve this problem. In our algorithm, the arm is represented by each feasible solution.

Theorem 2. *There exist I^N feasible solutions in a network topology with I data centers and N newly added VNF instances.*

We propose the Upper Bound Confidence (UCB) based VNF Placement algorithm. The main idea of the algorithm is not only considering the reward but also taking into account the size of the one-sided confidence interval for the average reward. Specifically, the result is denoted by policy index, which consists of current average reward and confidence interval. Confidence interval is a positive number between (0,1), which indicates the degree of exploration. The more we explore, the more accurate information we can get. In this way, we can exploit optimal solution and explore random solutions simultaneously so as to reduce the impact of noise. The process of UCB based VNF Placement algorithm is as follow: We first compute reward and confidence interval for each arm. Here we define the reward of arm j as $r_{j,t} = 1 - \frac{x^{(t)}_{p,m,i,m',i'} + l^{(t)}_{i,i'}}{band_{ii'}}$. After all arms have been selected, we will choose the arm with the maximum value in the next round.

The deployment algorithm can be described as follow, as shown in Algorithm 1. Algorithm 1 can be divided into two parts. The first part attempts at all feasible solutions (*lines* 4 – 12), and the second part constantly selects the arm with optimal value until the network is stable (*lines* 13 – 32). In each time slot, if $\Delta\rho_m^{(t)} > 0$, i.e., new demands are generated, we first deploy each of the solution in X, here X denotes the set of feasible deployment solutions (*lines* 3). Due to the constraints of CPU and memory, we need to determine whether the data center has enough resources to add new VNF instances. If so, Algorithm 3 is used to route the request

Algorithm 1 VNF placement based on UCB.

```
Input: $\Delta\rho_m^{(t)}$, $G(V,E)$, flow $f_p = (s_p, d_p, b_p, sc_p)$, $w_i^r (i \in I, r \in R)$, $M$
Output: $\rho_{m,i}^{(t)}$
 1: if the system is in provisioning time then
 2:   if $\Delta\rho_m^{(t)} > 0$ then
 3:     Follow theorem 2 and $\Delta\rho_m^{(t)}$ to compute all feasible deployment solutions $X = \{x_1, x_2, \ldots x_n\}$ as candidate arms;
 4:     for i = {1,2,3,...,n} do
 5:       Choose deployment solution $x_i$;
 6:       if $\sum_{m\in[M]} w_{m,i}^r + \sum_{\Delta\rho_m^{(t)}\in[M]} w_{m,i}^r > w_i^r$ then
 7:         Remove $x_i$ from solution set $X$;
 8:         Continue;
 9:       else
10:         Apply Algorithm 3 routing the flow $f_p$ in $G$ and compute policy index with $\frac{\sum_{t=1}^{T} r_{i,t}}{T} + \sqrt{\frac{2lnT}{n_i(T)}}$ for arm $i$;
11:       end if
12:     end for
13:     Create a list Y for all arms in descending order of policy index and record the sequence number of the first element as 1 ($Y_1$ is the arm with the largest policy index);
14:     ArmChanged ← True;
15:     i ← 1;
16:     while ArmChanged do
17:       Choose arm $Y_i$;
18:       if $\sum_{m\in[M]} w_{m,i}^r + \sum_{\Delta\rho_m^{(t)}\in[M]} w_{m,i}^r > w_i^r$ then
19:         i = i + 1;
20:         Continue;
21:       else
22:         Apply Algorithm 3 routing the request and compute the policy index of $Y_i$;
23:         Update the list Y in descending order of policy index;
24:         if The first element of Y has not changed then
25:           ArmChanged ← False;
26:           $\rho_{m,i}^{(t)}$ = $Y_1$ ;
27:         else
28:           i = 1;
29:         end if
30:       end if
31:     end while
32:     return $\rho_{m,i}^{(t)}$;
33:   end if
34: end if
```

and calculate the policy index. Otherwise, solution x_i is removed from the set X (*lines* 4 – 12). In the second part, we create a list Y in descending order of policy index and set the flag as True (*lines* 13 – 14). In the loop, a similar process is repeated to determine whether the data center has enough resources. If not, a suboptimal solution is selected and re-enter the loop. Otherwise, we compute the policy index and update set Y until the network is stable (*lines* 15 – 32). Here we define the confidence interval $c_{T,n_j(T)}$ as $\sqrt{\frac{2lnT}{n_j(T)}}$, where $n_j(T)$ is the number of times that arm j has been selected during time T.

4.3. Analysis of VNF placement algorithm

In our problem, service providers should choose an arm j (represented by solution x_j from set X), and each choice returns a reward $r_{j,1}, r_{j,2}, \ldots, r_{j,n}$, which obeys the random distribution with an unknown expectation. We measure *regret* to evaluate the performance of the VNF placement algorithm. The *regret* is caused by the fact that we cannot always choose the best solution, which is defined as

$$\mu^{\max}T - \mu_i \sum_{i=1}^{n} \mathbb{E}[n_i(T)] \tag{16}$$

where $\mu^{\max}$ is defined as $\max_{1\le j\le n} \mu_j$, and $\mathbb{E}[\cdot]$ denotes expectation.

Clearly, we can easily get $\sum_{j=1}^{n} n_j(T) = T$. We introduce $\Delta_j = \mu^{\max} - \mu_j$. Thus (13) can be rewritten as $\sum_{i=1}^{n} \Delta_i \mathbb{E}[n_i(T)]$, and the *regret* can be bounded by simply bounding each $\mathbb{E}[n_i(T)]$. We start from standard exponential inequalities to bound *regret*.

Lemma 1 (Chernoff-Hoeffding bound)**.** *Let $X_1, \ldots, X_n$ be random variables with common range [0,1] such that $\mathbb{E}[X_t] = \mu$. Let $S_n = X_1 + \cdots + X_n$. Then for all $a \ge 0$, we can get*

$$P\{S_n \ge n\mu + a\} \le e^{-2a^{2/n}}, P\{S_n \le n\mu - a\} \le e^{-2a^{2/n}} \tag{17}$$

Recall that the policy index is equal to confidence interval $c_{T,n_i(t)}$ add average reward $\bar{r}_{j,t}$. We introduce an indicator $a_{j,t} = 1$ if arm j is selected in time t and rewrite $n_j(t)$, $\bar{r}_{j,t}$ as $n^{\max}(t)$ and $\bar{r}_t^{\max}$ if solution j is the index of the optimal arm. Then, we can get the following theorem.

Theorem 3. *For $n > 1$, the regret of our algorithm after any number T of choices is no more than* $[8\sum_{i=1}^{n}(\frac{\ln T}{\Delta_i})] + (1 + \frac{\pi^2}{3})(\sum_{i=1}^{n}\Delta_i)$

Proof. Let $\mathcal{L}$ be a positive integer and in each time slot $1 \le \mathcal{L} \le n_j(t-1)$ we can get

$$\begin{aligned} n_j^{(T)} &= 1 + \sum_{t=n+1}^{T} \{a_{j,t}\} \\ &\le \mathcal{L} + \sum_{t=n+1}^{T} \left\{a_{j,t}\right\} \\ &\le \mathcal{L} + \sum_{t=n+1}^{T} \left\{\bar{r}^{\max}_{n^{\max}(t-1)} + c_{t-1,n^{\max}(t-1)} \le \bar{r}_{j,n_j(t-1)} + c_{t-1,n_j(t-1)}\right\} \\ &\le \mathcal{L} + \sum_{t=n+1}^{T} \left\{\min_{1\le p\le t-1} \bar{r}_p^{\max} + c_{t-1,p} \le \max_{\mathcal{L}\le p_j\le t-1} \bar{r}_{j,p_j} + c_{t-1,p_j}\right\} \\ &\le \mathcal{L} + \sum_{t=1}^{\infty}\sum_{p=1}^{t-1}\sum_{p_j=\mathcal{L}}^{t-1} \left\{\bar{r}_p^{\max} + c_{t,p} \le \bar{r}_{j,p_j} + c_{t,p_j}\right\} \end{aligned} \tag{18}$$

From $\bar{r}_p^{\max} + c_{t,p} \le \bar{r}_{j,p_j} + c_{t,p_j}$ we can obtain the following inequality

$$\bar{r}_p^{\max} \le \mu^{\max} - c_{t,p} \tag{19}$$

$$\bar{r}_{j,p_j} \geq \mu_j + c_{t,p_j} \tag{20}$$

$$\mu^* \geq \mu_j + 2c_{t,p_j} \tag{21}$$

Bounding the probability of (19) and (20) by Lemma 1, we have

$$P\{\bar{r}_p^{\max} \leq \mu^{\max} - c_{t,p}\} \leq e^{-4\ln t} = t^{-4} \tag{22}$$

$$P\{\bar{r}_{j,p_j} \geq \mu_j + c_{t,p_j}\} \leq e^{-4\ln t} = t^{-4} \tag{23}$$

According to (21), we can get $p_j \geq \lceil (8\ln T)/\Delta_j^2 \rceil$ and the (18) can be modified as

$$\begin{aligned}\mathbb{E}[n_j(T)] &\leq \lceil (8\ln T)/\Delta_j^2 \rceil + \sum_{t=1}^{\infty} \sum_{p=1}^{t-1} \sum_{p_j=\lceil (8\ln T)/\Delta_j^2 \rceil}^{t-1} \times (2t^{-4}) \\ &\leq \lceil (8\ln T)/\Delta_j^2 \rceil + \sum_{t=1}^{\infty} \sum_{p=1}^{t-1} \sum_{p_j=1}^{t-1} \times (2t^{-4}) \\ &\leq (8\ln T)/\Delta_j^2 + 1 + \frac{\pi^2}{3}\end{aligned} \tag{24}$$

This completes the proof. □

5. Online routing of minimum-cost algorithm with delay constraints

As aforementioned, we have known the locations of VNF instances. In this section, we would like to propose an online routing algorithm for each flow with minimum cost while satisfying delay constraints by constructing a multi-level acyclic graph.

5.1. Construction of multi-level acyclic graph

Given a flow p with a service chain $sc_p = \{sc_{p,1}, sc_{p,2}, \ldots, sc_{p,M}\}$, we define $|sc_p| = M$ and use $G(L_p, N_p)$ to denote the multi-level acyclic graph with $M+2$ levels. The first level l_1 represents the source and the last level l_{M+2} represents the destination. These two levels contain only one node, s_p and d_p, respectively. Level l_i $(\forall i \in \{2, \ldots, (M+1)\})$ represents the set of candidate data centers that can process $(i-1)$-th network function. We can define $L_p = \bigcup_{i=2}^{M+1} l_i \bigcup \{s_p, d_p\}$ and use $l_{i,j} (j = \{1, 2, 3 \ldots \rho_m^{(t)}\}, \forall m \in M)$ to indicate the j-th candidate data center in level i.

For each node in level l_i and l_{i+1}, we use directed edges to connect all pairs of nodes and use (l_i, l_{i+1}) to denote the set of all edges between level l_i and level l_{i+1}. Each edge is the shortest path in $G(V, E)$ if such a path exist. In general, N_p can be represented as $\{(s_p, l_1)\} \bigcup \{(l_{M+1}, d_p)\} \bigcup_{i=2}^{M} \{(l_i, l_{i+1})\}$. Fig. 2 gives an example of multi-level acyclic graph.

5.2. Heuristic algorithm with delay constraint

It is known from Ref. [37] that the delay-constrained shortest path problem is NP-hard. In the following, we will propose a heuristic algorithm to solve the delay-constrained shortest path problem.

Given a flow f_p in $G(V, E)$ with a sequence of network functions in a specified order, the purpose of our algorithm is to find the shortest path from source to destination with the minimum cost while meeting delay constraint. We first construct multi-level acyclic graph $G(L_p, N_p)$. The edge in acyclic graph is the shortest path in $G(V, E)$. Then function *SettleInPath* is called to find the shortest path by the weight $\gamma_{p,i,i'}$. If the path meets the delay constraint, this path is regarded as the optimal solution and the algorithm stops (*lines* 1 – 5). Otherwise, we change

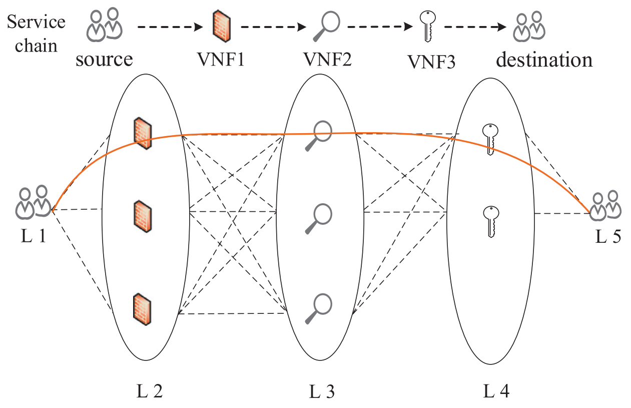


Fig. 2. An example of a multi-level acyclic graph, where l_2, l_3, l_4 represent the set of candidate data centers.

the weight to $\varphi_{p,i,i'}$ and call function *SettleInPath* to find the path with the lowest latency. If the delay of the obtained path is less than the threshold, we regard it as the appropriate path we have founded so far. If not, there is no suitable path that can meet the delay constraint and the network cannot process this flow (*lines* 6 – 9). To find the path with the lowest cost, we adopt Lagrange relaxation to construct the multiplier λ to increase the dominance of H_d. The λ is calculated by the current information of path H_c and H_d (*lines* 10 – 12). Same as before, we call function *SettleInPath* to find the path H_r by the new weight $\gamma_{p,i,i'} + \lambda \cdot \varphi_{p,i,i'}$. If $c_\lambda(H_r) + c'(H_r) = c_\lambda(H_c) + c'(H_c)$, then the H_c and H_d are minimum cost paths. Otherwise, we replace either H_d or H_c with H_r according to whether H_r meets the delay constraint and repeat this step until $c_\lambda(H_r) + c'(H_r) = c_\lambda(H_c) + c'(H_c)$ (*lines* 13 – 24).

The function *SettleInPath* ensures that the shortest path can be found while satisfying the resource constraint. Parameter *type* is a binary variable equals to *cost* or *delay*. In the function, we first set the Flag to *False* and then determine the value of *type*. According to *type*, we find a path H with minimal cost or delay (*lines* 27 – 35). Notice that H is composed of many logical links in the graph. If one of the paths does not have enough bandwidth, or physical machines on the path does not have enough resources (CPU, memory), we will remove it from the graph and redirect the flow to another path capable of sufficient resources (*lines* 36 – 42). If all the links in the graph have been removed and no H has been found, the function will terminate (*lines* 43 – 45).

In the following we will analyze the time complexity of Algorithm 2. The first part of the algorithm is the construction of multi-level acyclic graph. We can divide the construction into two step. Firstly we need to take $O(\sum_{i=1}^{l+1} |u_i| + 2)$ to construct each level in L_p, and secondly take $O(\sum_{i=1}^{l+1} (|u_i| \times |u_{i+1}|))$ to connect each level in N_p, which equals to $O(|u|^3)$. According to [38], the complexity of lagrange relaxation is $O(|u|^2 log^4 |u|)$. The function *SettleInPath* takes $O(|u|^2)$ time. Thus, the time complexity of Algorithm 2 is $O(|u|^3)$.

In the final, we will summarize our complete algorithm in Algorithm 3. Algorithm 3 incorporates the Fourier-Series-based forecast method to get the number of newly VNF instances, together with VNF deployment and routing algorithm. Here, the forecast algorithm is executed in prediction stage to predict whether it is necessary to update the network topology and return the number of newly VNF instances. Deployment algorithm uses the number of VNFs as input to determine the locations of each VNF instance. Finally, we employ the proposed routing algorithm to choose the shortest path for each flow with delay and resources constraints.

Algorithm 2 Heuristic algorithm for finding shortest path with delay constraint.

```
Input: $G(V, E)$, flow $f_p = (s_p, d_p, b_p, sc_p)$, $d^{(t)}$, $M$, $Band_{path}$, UnSolvedPath = 0
Output: Shortest path in $G$ with delay constraint
1: construct multi-level acyclic graph represented by $G(L_p, N_p)$;
2: $H_c$ = SettleInPath($G(L_p, N_p)$, $f_p$, $\gamma_{p,i,i'}$, cost);
3: if $d(H_c) + d_r^{(t)} \leq d^{(t)}$ then
4:     return $H_c$;
5: else
6:     $H_d$ = SettleInPath($G(L_p, N_p)$, $f_p$, $\varphi_{p,i,i'}$, delay);
7:     if $d(H_d) + d_r^{(t)} > d^{(t)}$ then
8:         UnSolvedPath = UnSolvedPath + 1;
9:     else
10:        $boolean \leftarrow false$;
11:        while $boolean : boolean = true$ do
12:            Construct the multiplier $\lambda = \frac{c(H_c)-c(H_d)}{d(H_d)-d(H_c)}$, where $c(H)$ and $d(H)$ are the cost and delay on edges;
13:            $H_c$ = SettleInPath($G(L_p, N_p)$, $f_p$, $\gamma_{p,i,i'} + \lambda \cdot \varphi_{p,i,i'}$, cost);
14:            if $c_\lambda(H_r) + c'(H_r) = c_\lambda(H_c) + c'(H_c)$ then
15:                $boolean \leftarrow true$;
16:            else
17:                if $d(H_r) + d_r^{(t)} \leq d^{(t)}$ then
18:                    $H_d \leftarrow H_r$;
19:                else
20:                    $H_c \leftarrow H_r$;
21:                end if
22:            end if
23:        end while
24:        return The shortest path $H_d$;
25:    end if
26: end if
27: function SettleInPath($G \leftarrow graph$, $p \leftarrow flow$, $\omega \leftarrow weight$, $type \leftarrow cost$ or $delay$)
28: Flag ← True;
29: while Flag do
30:     Flag ← False;
31:     if $type$ == $cost$ then
32:         Apply Dijkstra's algorithm to find the shortest path $H$ for $p$ with the minimum cost in $G$ by the weight $\omega$;
33:     else
34:         Apply Dijkstra's algorithm to find the shortest path $H$ for $p$ with the minimum delay in $G$ by the weight $\omega$;
35:     end if
36:     for $path$ in $H$ do
37:         if $b_p > Band_{path}$ or $F_{p,i,i'} > w_i^r - \sum_{m\in[M]} w_{m,i}^r$ then
38:             Remove the path from $G$;
39:             Flag ← True;
40:             Break;
41:         end if
42:     end for
43:     if $G$ is $\emptyset$ then
44:         return UnSolvedPath + 1;
45:     end if
46: end while
47: return H
```

Algorithm 3 The complete algorithm.

```
Input: $\rho_m^{(t-1)}$, $\Delta\rho_m^{(t)}$, $G(V, E)$, flow $f_p = (s_p, d_p, b_p, sc_p)$
Output: $\rho_{m,i}^{(t)}$, shortest path in $G(V, E)$
1: while The network is operational do
2:     if The system is in prediction stage then
3:         Apply Forecasting Algorithm to get $\Delta\rho_m^{(t)}$;
4:     end if
5:     if The system is in provisioning time then
6:         Apply Deployment Algorithm to get $\rho_{m,i}^{(t)}$;
7:         Apply Routing Algorithm to find the shortest path in $G(V, E)$;
8:     end if
9: end while
```

6. Performance evaluation

In this section, we present the experiment environment and results of the performance evaluation.

6.1. Simulation setup

We conduct our simulations on the Fat-Tree [39], VL2 [40] and one inter-datacenter network. Each data center network consists of 128 severs [16]. Each sever is connected to some edge switches, the number of which may vary according to different topologies. The bandwidth of links between nodes is 1000*Mbps* and the CPU and memory in sever are 4*10,000 MIPS and 4GB respectively [8,30]. For inter-datacenter network, there are 100 access nodes with 150 links and 20 data centers [30]. The bandwidth of links between nodes is 10000*Mbps* and the CPU and memory in severs are 12*10,000 MIPS and 12GB respectively [8,30]. What's more, the delay between data centers are set proportional to their geographic distances. We assume the distance between two data centers is from 500 to 1000 KM [11] and introduce a multiplier to calculate the delay. Following [16], the capability of each VM is limited by its CPU. We assume the CPU of each VM is 10,000 MIPS and the memory is 1GB.

In this paper, six types of network functions are considered, i.e., firewall, NAT, IDS, Load Balancing, gateways and Proxy [11,13]. The size of CPU and memory requirement for a VNF are chosen from an exponential distribution with mean 2000 MIPS and 0.5GB [41]. Following [5,24] configurations, the change ratio of firewall and IDS is 0.9 and 0.8, respectively. We assume the cost of the link is inversely proportional to the link capacity, which means that the more bandwidth resources, the lower cost caused by the transmission of unit-sized flow. In each time slot, we generate 2000 flows with bandwidth demand between [10,60] Mbps for data center networks and 4000 flows with bandwidth demand between [100,150] Mbps for the inter-datacenter network. In order to verify the performance of our algorithm in actual network experimental environment, the flows are generated based on the Alibabas cluster trace [42]. The cluster trace is obtained by about 1300 machines during a period of 12 hours. The trace contains the collocation of online services (aka long running applications) and batch workloads. We set the length of a timeslot as 20 minutes so that the network has enough time to update.

We compare our method with the following algorithms.

- **Greedy Algorithm** [12]: It adopts an online algorithm to minimize the cost for scaling VNF instances, while in the meantime, minimizing the congestion in face of unknown varying traffic.
- **Elastic SFC Algorithm (ESFC)** [30]: It adopts horizontal and vertical scaling of VNFs and dynamic bandwidth allocation to optimize network performance.
- **Cross Rack Pipelined SFC Algorithm (CRPS)** [31]: It first predicts the upper bound of traffic rate in the next time slot and then scales VNF instances in order to keep the high availability of service.

For the background traffic, we send 200*Mbps* traffic in expectation varying from 100*Mbps* to 500*Mbps* on each link. Equation (25) is the probability density function that ensures the expectation of background traffic c in the range $[a, b]$.

$$p(t) = \begin{cases} \frac{b-a}{bc-ab} & a \leq t < c \\ \frac{a}{b^2-cb} & c \leq t \leq b \end{cases} \tag{25}$$

6.2. Performance comparison

In this section, we compare our algorithm with others under multiple network topologies.

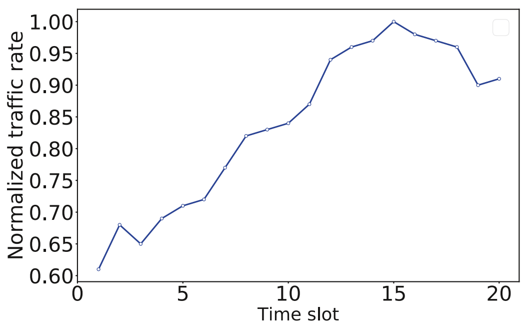


Fig. 3. The evolution of Normalized traffic rates.

6.2.1. Estimation of traffic

In this section, we compare the performance of our prediction algorithm with the CRPS algorithm in [31] in terms of traffic estimation. The estimation error is defined as

$$\Delta = 1 - \frac{\sum_{i=1}^{n} Y(i) - \hat{Y}(i)}{\sum_{i=1}^{n} Y(i) - \overline{Y}} \tag{26}$$

where $\hat{Y}(i)$ is the estimated value, $\overline{Y}$ is the average of training set and $Y(i)$ is the real value. We select a part of data from Alibabas cluster trace [42] as the training set, which is shown in Fig. 3. By the evolution of traffic rate in Fig. 3, we can define the trend function as $b_0 ln(t)$. We use Statistical Product and Service Solutions (SPSS) to solve the regression function and consider the impact of changes in the fitting order (m in the formula (12)) and the training set size (n in the formula (26)) on performance.

Fig. 4(a) shows how the error changes with fitting order m. We fix the training set size to be 3 (data generated in 3 time slots) and change the fitting order. We can see from Fig. 4(a) that as the fitting order increases, the error is reduced. When the fitting order reaches 6, the system is basically stable and our algorithm can achieve better performance. This is because we introduce the trend function. Since the traffic characteristic in network changes from time to time, the coefficient may not work in a certain situations. Thus in our algorithm, we introduce an appropriate trend function to improve the robustness and estimation accuracy.

In Fig. 4(b), we fix the fitting order as 5 and change the size of training set n. The curve in the figure shows that the size of the training set has little effect on the accuracy. However, there is an interesting phenomenon that the error first reduces and then increases. This is due to that with the increase of the data set, the traffic characteristics become complicated, and history information may play negative role to train our model. By experimental results, we find that the method can derive the best performance when we set the fitting order as 6, the train set as 4 and the trend function as $ln(t)$.

6.2.2. Resource usage and latency

In this section, we compare resource usage and latency of our algorithms with those of other three algorithms, i.e., Greedy algorithm [12], ESFC algorithm [30] and CRPS algorithm [31] under three network topologies. We first check the impact of the amount of data on resource usage and latency, i.e., total cost, as shown in Fig. 5. We can observe from Fig. 5 that our algorithm can save more than 20% cost against others. This is because our algorithm first predicts whether there is a demand to add new VNF instances, which avoids frequent changes to the network topology, and adopts the UCB algorithm to deploy VNF instances, which can avoid link congestion and reduce transmission cost. However, in the ESFC, it first allocates the residual resources in the network according to end-to-end delay, and then scales the network vertically or horizontally. Nevertheless, deploying VNF instances based on resources can easily cause link congestion and high transmission cost. In $\epsilon_t - greedy$ algorithm and CRPS, the frequent changes to network topology can significantly increase the cost and the traffic is concentrated on some data centers, resulting in high processing cost. It is worth noting that in Fig. 5(c), the total cost of ESFC is greater than CRPS when the training set size reaches 5000. This is because ESFC can make full use of resources in the network. When the training set is large, data centers and links are fully loaded. Therefore, additional VNF instances need to be added, resulting in high cost.

We further check the impact of the length of the service chain. We test four ranges of service chains, which are (1,2), (3,4), (5,6) and (7,8) respectively [16]. Fig. 6 shows the longer service chains causes more cost and our algorithm can also achieve a good performance against others. This is because our algorithm can update topology as needed and the distributed data centers can provide better routing environment. What's more, we introduce confidence interval $c_{T,n_j(t)}$ in UCB algorithm to consider the uncertainty of the reward, and the candidates can get more opportunities to be selected. In CRPS and ESFC, the longer service chains causes more cost and ESFC can get better performance in inter-datacenter network. This is because our network is a cross rack pipelined model and there are always some redundant VMs [31]. ESFC can allocate the residual resources well and scale the network vertically. Notice that in $\epsilon_t - greedy$ algorithm, the length of the service chain has little effect on the cost. This is because in this simulation, the network size is relatively small, the construction cost of VNF instance is a major part of the total cost.

For the same reason, we can get the algorithm running time in Fig. 7 and our algorithm can also achieve good performance. However, in Fig. 7(b), when the length of service chain reaches [7,8], the performance of CRPS is better than ESFC. This is because the more requests consume the limited bandwidth especially after the residual resources have been allocated. The fixed amount of net-

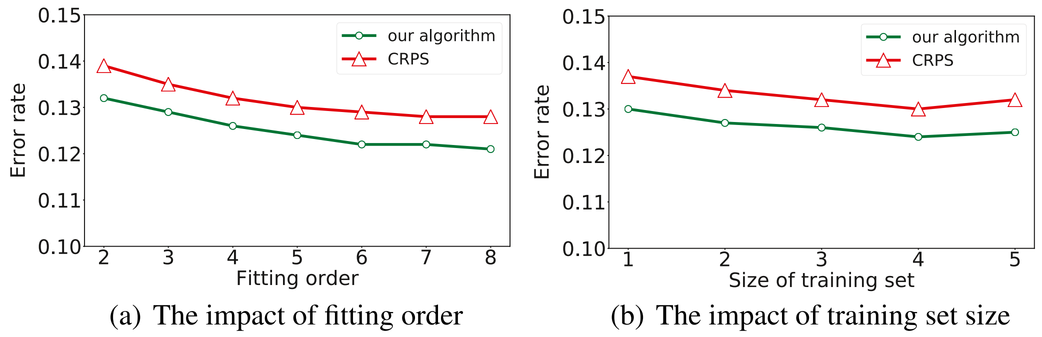


Fig. 4. The evolution of prediction errors with fitting order and training set size.

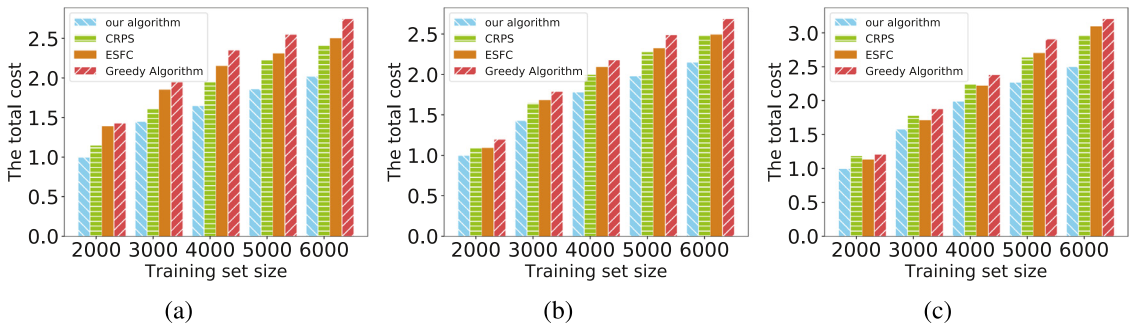


Fig. 5. The impact of the amount of data set on total cost in (a) Fat-Tree, (b) VL2, and (c) inter-datacenter network.

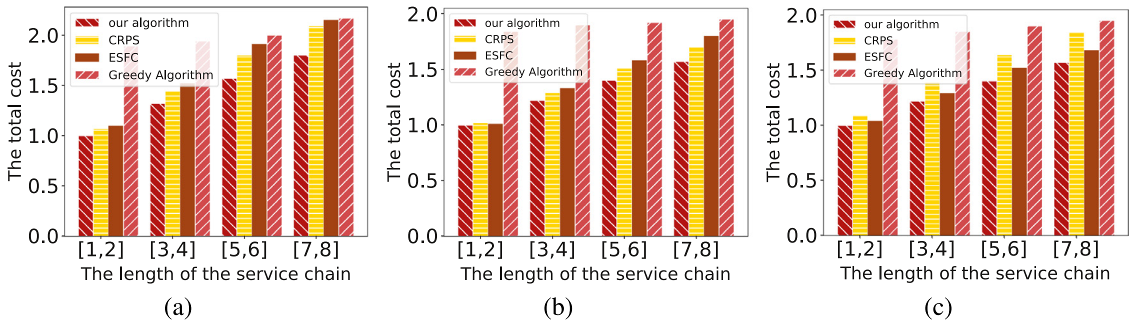


Fig. 6. The impact of the length of service chain on total cost in (a) Fat-Tree, (b) VL2, and (c) inter-datacenter network.

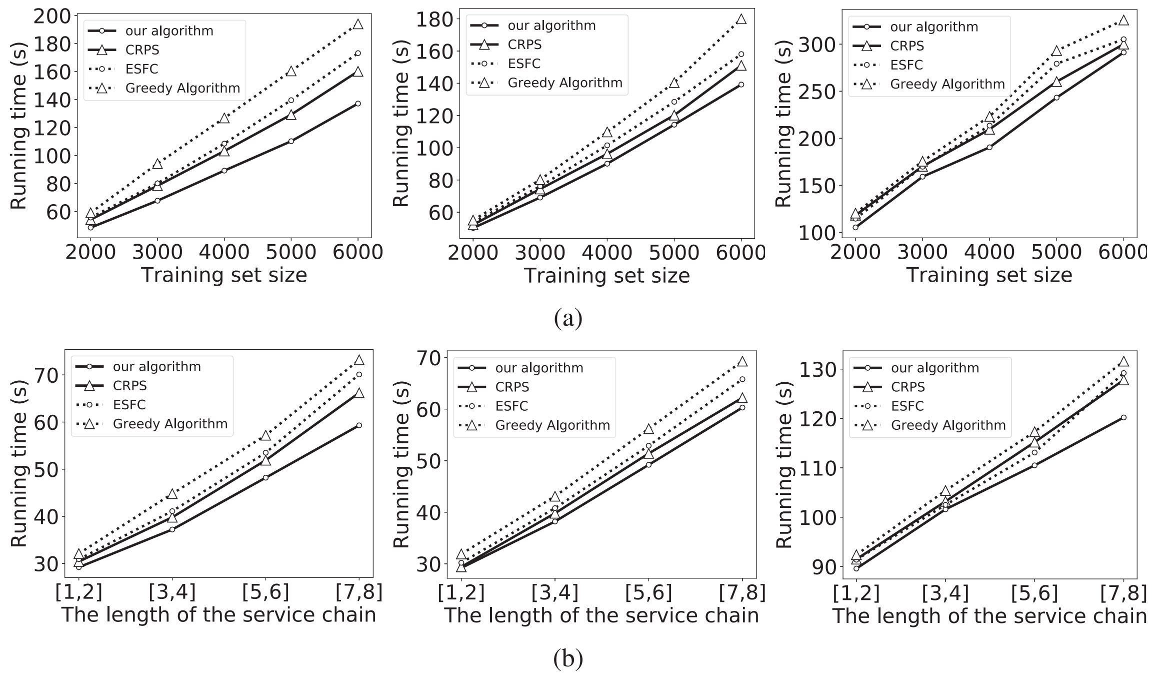


Fig. 7. (a) The impact of the amount of training set on algorithm running time under three topologies and (b) the impact of the length of the service chain on algorithm running time under three topologies. The three topologies from left to right is Fat-Tree, VL2, and inter-datacenter network, respectively.

work resources with more VNFs cause the bottleneck in network transmission, which results in running time increasing.

Finally, we will consider the impact of end-to-end delay constraint on the total cost of the algorithm as shown in Fig. 8, where the delay is considered from 40 ms to 200 ms [13]. We can observe From Fig. 8 that with the relaxation of delay constraint, the cost is reduced and can be stable when the delay constraint reaches a certain value. This is because when the delay constraint is relaxed, we do not need to discard some low-cost paths that violate the constraint. If the delay constraint exceeds the latency on the minimum-cost path, the cost will remain stable. In addition, we can find our algorithm can achieve better performance than others. This is due to that by the multiplier λ in Algorithm 2, we can constantly change the link weight to find the minimum cost path that satisfies the delay constraint. However, CRPS algorithm and greedy algorithm only assign the traffic based on the residual resources and ignore the dynamic changes in link cost. We can see From 8(c) that when the constraint of end-to-end delay reaches 120ms, CRPS can achieve better performance than ESFC. This is because ESFC allocates the residual resources in the network based on end-to-end

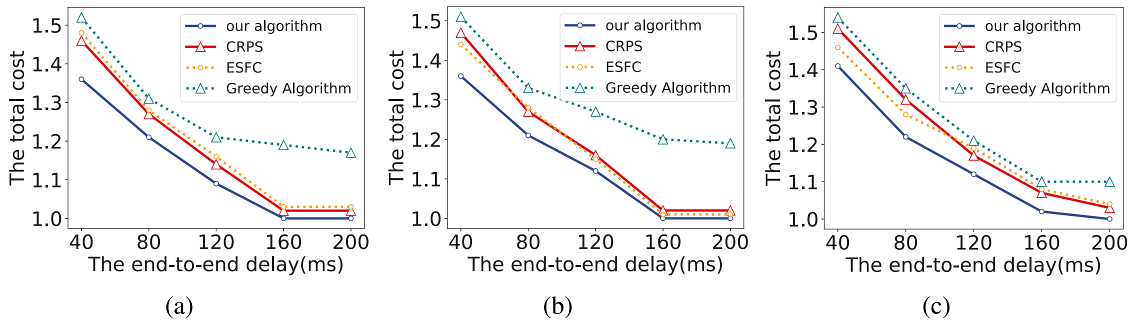


Fig. 8. The impact of the delay constraint on total cost in (a) Fat-Tree, (b) VL2, and (c) inter-datacenter network.

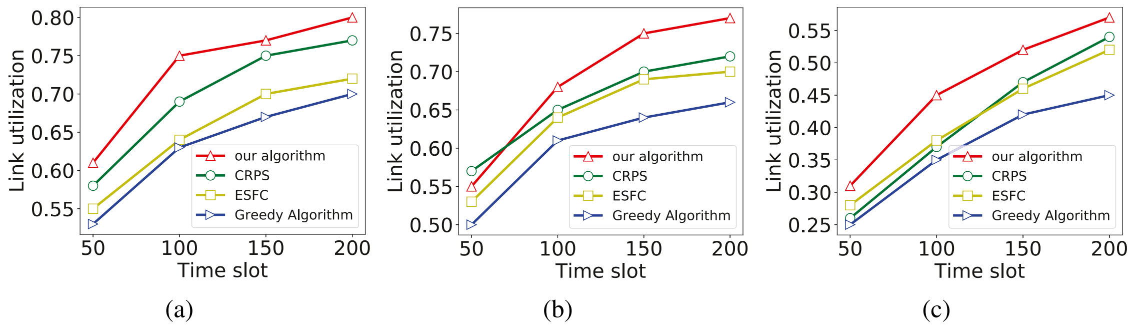


Fig. 9. The change of link utilization on (a) Fat-Tree, (b) VL2, and (c) inter-datacenter network.

delay. When delay constraint is relaxed, more low-cost but high-delay paths can be found. In this case, resources allocation methods in ESFC may not work well and cause high cost.

6.2.3. Performance of link utilization

In this experiment, we define the link utilization as the number of the used links divided by the total number of links. Fig. 9 shows the trend of link utilization as time slots increase. We can observe that as the system operating continues, the link utilization in the network is gradually increasing. The reason is that the network topology is a stochastic model and each switch can serve as the source and destination of the flow, which cause many non-repeating routing paths. Compared to other three algorithms, $\epsilon_t - greedy$ has the lowest link utilization, followed by ESFC. This is because the limitations of VNF deployment in ESFC may cause flows to be only processed in one data centers.

In $\epsilon_t - greedy$ algorithm, the low level of exploration of candidate solutions results in the insufficient use of links. And CRPS algorithm only considers the residual capacity of links and wants to fully utilize the capacity of data centers, which may result in data transmission being concentrated on a small number of links. There is an interesting observation in Fig. 9(b) that CRPS has better performance than proposed algorithm when time slot is equal to 60. This is due to the exploration mechanism of feasible solutions. When time slot is equal to 60, the confidence interval is small and the exploration of other feasible solutions is insufficient, which leads to the concentration of data transmission on some links.

6.2.4. Performance of congestion

We evaluate the reward $r_{j,t}$ achieved by $\epsilon_t - greedy$ algorithm and ours under two types of background traffic. One is uniform and the other is permutation [28]. For uniform background traffic, data centers send 500*Mbps* traffic in expectation varying from 300*Mbps* to 1000*Mbps* on each link. For permutation background traffic, data centers randomly choose a part of links and generate 1200*Mbps* traffic in expectation between 800*Mbps* and 3000*Mbps*.

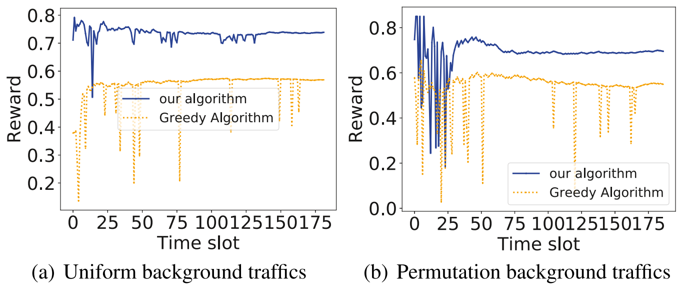


Fig. 10. The evolution of rewards in (a) uniform background traffics and (b) permutation background traffics.

As defined before, the reward indicates the degree of the congestion on the link.

Fig. 10 shows that the reward varies as the growth of time slots. Obviously, the trend is gradually approaching a fixed value. This is because as time slots grow, the exploration of candidates is gradually reduced. The value of $\epsilon_t - greedy$ algorithm and ours is 0.56, 0.73 respectively in uniform background traffic and 0.54, 0.69 respectively in permutation background traffic. The reason is that our algorithms consider the uncertainty of the reward, and the introduction of confidence interval makes candidates get more opportunities to be selected. In $\epsilon_t - greedy$ algorithm, the exploration is random and has no any distinction, which may lead to low reward in long-term processing. What's more, due to the error in exploring the feasible solutions, the curve in Fig. 10(b) fluctuates greatly. But the system will return a better solution and eventually approach a stable value.

7. Conclusions

In this paper, we consider the horizontal scaling of VNFs. Firstly, we construct the system model and formulate a cost minimization problem. In practice, the network is a time-varying system and network topology needs to be constantly adjusted by adding or removing VNF instances. A good way to reduce cost is to avoid frequent updates to the network topology. Thus, we adopt a forecasting algorithm to update network on-demand. Furthermore, due to the sharing nature of data centers, there may exist some other workloads to affect the performance of system, we propose a deployment algorithm based on online-learning to eliminate the influence. In this algorithm, the performance of system is characterized as reward and the optimal deployment solution can be finally found out. When we have configured the network, we propose a routing algorithm to guarantee that the service chain is processed in a specific order while meeting the delay constraint. Finally, experimental results show that our algorithms can reduce more than 20% cost while maintain system performance compared to other heuristic algorithms.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

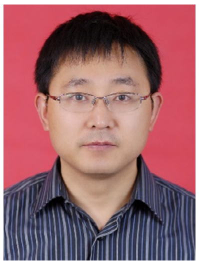
Acknowledgments

This work was supported by the National Key R&D Program of China (2018YFB0803400), National Natural Science Foundation of China (no. 61772432, 61772433), the Technological Innovation and Application Demonstration Projects of Chongqing (cstc2018jszx-cyztzxX0014), the Fundamental Research Funds for the Central Universities (2019CDYGZD004).

References

[1] A. Gupta, M.F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, B. Mukherjee, On service-chaining strategies using virtual network functions in operator networks, Comput. Netw. 133 (2016) 1–16.
[2] A. R. D. Clarke, P. Willis, Network functions virtualization, 2012, https://portal.etsi.org/portal/server.pt/community/NFV/367.
[3] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, P. Demeester, Network service chaining with optimized network function embedding supporting service decompositions, Comput. Netw. 93 (2015) 492–505.
[4] P. Quinn, T. Nadeau, Service function chaining problem statement, 2013, https://datatracker.ietf.org/doc/rfc7498/.
[5] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, A. Akella, Opennf: enabling innovation in network function control, in: ACM Conference on SIGCOMM, 2014, pp. 163–174.
[6] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, F. Lau, Online vnf scaling in datacenters, in: 2016&nbsp;IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 140–147.
[7] T. Benson, A. Anand, A. Akella, M. Zhang, Understanding data center traffic characteristics, in: Computer Communication Review - CCR, 2009, pp. 65–72.
[8] J. Duan, C. Wu, F. Le, A.X. Liu, Y. Peng, Dynamic scaling of virtualized, distributed service chains: a case study of ims, IEEE J. Sel. Areas Commun. (2017) 2501–2511.
[9] M. Rost, S. Schmid, It's a match!: Near-optimal and incremental middlebox deployment, in: ACM SIGCOMM Computer Communication Review, 2016, pp. 30–36.
[10] Q. Sun, P. Lu, W. Lu, Z. Zhu, Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement, in: Global Communications Conference, 2017, pp. 1–6.
[11] Y. Jia, C. Wu, Z. Li, F. Le, A. Liu, Online scaling of nfv service chains across geo-distributed datacenters, IEEE/ACM Trans. Netw. 26 (2) (2018) 699–710.
[12] J. Liu, L. Yong, Z. Ying, S. Li, D. Jin, Improve service chaining performance with optimized middlebox placement, IEEE Trans. Serv. Comput. (2017) 560–573.
[13] M. Jia, W. Liang, M. Huang, Z. Xu, Y. Ma, Routing cost minimization and throughput maximization of nfv-enabled unicasting in software-defined networks, IEEE Trans. Netw. Serv.Manage. (2018) 732–745.
[14] H.D. Chantre, N.L.S.D. Fonseca, Multi-objective optimization for edge device placement and reliable broadcasting in 5g nfv-based small cell networks, IEEE J. Sel. Areas Commun. (2018) 2304–2317.
[15] M. Welzl, Network congestion control: Managing internet traffic, John Wiley and Sons, 2006, pp. 259–263. 05.
[16] T.W. Kuo, B.H. Liou, C.J. Lin, M.J. Tsai, Deploying chains of virtual network functions: On the relation between link and server usage, in: IEEE INFOCOM 2016 - the IEEE International Conference on Computer Communications, 2016, pp. 1–9.
[17] R. Cohen, L. Lewin-Eytan, J.S. Naor, D. Raz, Near optimal placement of virtual network functions, in: Computer Communications, 2015, pp. 1346–1354.
[18] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, L. Feng, Trajcompressor: an online map-matching-based trajectory compression framework leveraging vehicle heading direction and change, IEEE Trans. Intell. Transp.Syst. (2019) 1–17.
[19] C. Chen, Y. Ding, Z. Wang, J. Zhao, B. Guo, D. Zhang, Vtracer: when online vehicle trajectory compression meets mobile edge computing, IEEE Syst. J. (2019) 1–12.
[20] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, Clickos and the art of network function virtualization, in: Usenix Conference on Networked Systems Design and Implementation, 2014, pp. 459–473.
[21] J. Hwang, T. Wood, K. Ramakrishnan, Netvm: high performance and flexible networking using virtualization on commodity platforms, in: Usenix Conference on Networked Systems Design and Implementation, 2014, pp. 445–458.
[22] V. Sekar, N. Egi, S. Ratnasamy, M.K. Reiter, G. Shi, Design and implementation of a consolidated middlebox architecture, in: Usenix Conference on Networked Systems Design and Implementation, 2012, p. 24.
[23] H. Moens, F. De Turck, Vnf-p: a model for efficient placement of virtualized network functions, in: International Conference on Network and Service Management, 2014, pp. 418–423.
[24] M.F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, On orchestrating virtual network functions in nfv, 11th International Conference on Network and Service Management (CNSM) (2015) 50–56.
[25] C. Pham, N.H. Tran, S. Ren, W. Saad, C.S. Hong, Traffic-aware and energy-efficient vnf placement for service chaining: joint sampling and matching approach, IEEE Trans. Serv. Comput. (2017) 1.
[26] R. Cziva, C. Anagnostopoulos, D.P. Pezaros, Dynamic, latency-optimal vnf placement at the network edge, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 693–701.
[27] S. Agarwal, F. Malandrino, C.F. Chiasserini, S. De, Vnf placement and resource allocation for the support of vertical services in 5g networks, IEEE/ACM Transactions on Networking (2019) 433–446.
[28] X. Wang, C. Wu, F. Le, F.C.M. Lau, Online learning-assisted vnf service chain scaling with network uncertainties, in: IEEE International Conference on Cloud Computing, 2017, pp. 205–213.
[29] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, B. Mukherjee, Auto-scaling vnfs using machine learning to improve qos and reduce cost, in: 2018&nbsp;IEEE International Conference on Communications (ICC), 2018, pp. 1–6.
[30] A.N. Toosi, J. Son, Q. Chi, R. Buyya, Elasticsfc: auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds, J. Syst. Softw. (2019) 152.
[31] H. Tang, D. Zhou, D. Chen, Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers, IEEE Trans. Parallel Distrib.Syst. (2019) 152–167.
[32] J. Pei, P. Hong, K. Xue, D. Li, Resource aware routing for service function chains in sdn and nfv-enabled network, IEEE Trans. Serv. Comput. (1939) 1.
[33] H. Khedher, E. Abd-Elrahman, A. Kamal, H. Afifi, Opac: an optimal placement algorithm for virtual cdn, Comput. Netw. 120 (2017) 12–27.
[34] A. Gember, A. Krishnamurthy, S.S. John, R. Grandl, X. Gao, Ashok anand, theophilus benson, aditya akella, and vyas sekar. stratos: a network-aware orchestration layer for middleboxes in the cloud, 2013, Technical Report, arXiv:1305.0209.
[35] R. Cziva, S. Jouet, D.P. Pezaros, Roaming edge vnfs using glasgow network functions, in: Conference on ACM SIGCOMM 2016 Conference, 2016, pp. 601–602.
[36] M. Huang, W. Liang, Z. Xu, S. Guo, Efficient algorithms for throughput maximization in software-defined networks with consolidated middleboxes, IEEE Trans. Netw. Serv.Manage. (2017) 631–645.
[37] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, WH Freeman, 1979.
[38] A. Juttner, B. Szviatovski, I. Mecs, Z. Rajko, Lagrange relaxation based method for the qos routing problem, in: Infocom Twentieth Joint Conference of the IEEE Computer Communications Societies IEEE, 2001, pp. 859–868.
[39] R.N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: a scalable fault-tolerant layer 2 data center network fabric, in: 2009 ACM SIGCOMM, 2009, pp. 39–50.
[40] GREENBERG, Albert, HAMILTON, R. James, JAIN, KANDULA, Srikanth, KIM, Changhoon, Vl2: a scalable and flexible data center network, Commun. ACM (2009) 95–104.
[41] H. Yu, J. Yang, C. Fung, Elastic network service chain with fine-grained vertical scaling, in: 2018&nbsp;IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–7.
[42] H. Ding, Alibaba trace. https://github.com/alibaba/clusterdata.

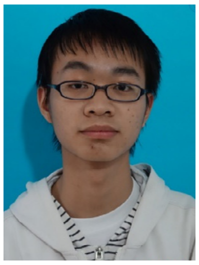
Yifu Yao received the B.S. degree in communication engineering from Southwest University, Chongqing, China, in 2017. He is currently working toward the MS degree in signal and information processing, Southwest University. His research interests include network functions virtualization and data center networks.



Songtao Guo received the B.S, M.S, and Ph.D degrees in computer software and theory from Chongqing University, Chongqing, China, in 1999, 2003, and 2008, respectively. He was a professor from 2011 to 2012 at Chongqing University and a professor from 2012 to 2018 at Southwest University. He is currently a full professor at Chongqing University, China. He was a senior research associate at the City University of Hong Kong from 2010 to 2011, and a visiting scholar at Stony Brook University, New York, from May 2011 to May 2012. His research interests include wireless networks, mobile cloud computing and parallel and distributed computing. He has published more than 100 scientific papers in leading refereed journals and conferences. He has received many research grants as a principal investigator from the National Science Foundation of China and Chongqing and the Postdoctoral Science Foundation of China.

Pan Li received the B.S. degree in Computer science and Engineering from Chongqing Three Gorges University, Chongqing, China, in 2016. He is currently working toward the Master degree in signal and information processing, Southwest University. His research interests include data center networks and software defined networking.

Guiyan Liu received the B.S. degree in telecommunications engineering from Southwest University, Chongqing, China, in 2014. She is currently working toward the PhD's degree in signal and information processing, Southwest University. Her research interests include stream scheduling in data center networks and software defined networking.



Yue Zeng received the B.S. degree in Computer science and Engineering from Chongqing Three Gorges University, Chongqing, China, in 2016 and Master degree in signal and information processing, Southwest University. He is currently working toward the Ph.D. degree in Computer science and Engineering, Nanjing University. His research interests include network energy saving and software defined networking.